UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

DETECCIÓN DE PERSONAS EN GRUPOS

Marta Villanueva Torres.
Tutor: Álvaro García Martín
Ponente: José M. Martínez Sánchez

-TRABAJO DE FIN DE GRADO-

Departamento de Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio 2015

DETECCIÓN DE PERSONAS EN GRUPOS.

Marta Villanueva Torres
Tutor: Álvaro García Martín
Ponente: José M. Martínez Sánchez

Video Processing and Understanding Lab
Departamento de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio 2015

Resumen

En los últimos años, el procesamiento de imagen y vídeo digital ha sufrido una constante evolución. Esto es debido a su relevancia en áreas tan importantes y actuales como la vídeo-seguridad, es por ello que la investigación en este campo se vuelve tan interesante de cara a innovar creando nuevas técnicas de análisis o mejorando las existentes.

Dentro del procesamiento de vídeo, la detección de personas es una de las tareas más complejas, a pesar de ello, actualmente, existen algoritmos que presentan resultados óptimos. Sin embargo, cuando la detección de personas se evalúa sobre entornos más complejos el rendimiento de dichos algoritmos se reduce notablemente, es por eso que este trabajo tiene como objetivo la implementación de un algoritmo de detección de personas en grupos en C++, así como su integración en una plataforma de análisis de vídeo denominada DiVA, que permite su ejecución sobre vídeos *off-line*, a fin de comprobar la funcionalidad y eficiencia del algoritmo en entornos de alta densidad.

El algoritmo implementado, conocido como *Multi-configuration Part-based Person Detector,* es una adaptación del algoritmo conocido como DTDP *(Discriminatively Trained Part Based Models)* para mejorar la detección en entornos con mayor densidad de personas. Basa sus detecciones en una búsqueda exhaustiva, en la que utiliza como referencia modelos de persona formados por la mezcla de diferentes partes del cuerpo. Gracias a esta búsqueda exhaustiva el algoritmo obtiene buenos resultados a costa de un mayor tiempo de procesamiento, es por eso que el análisis de este algoritmo no es posible en tiempo real.

Para completar nuestro objetivo se ha desarrollado una interfaz gráfica que permite la interacción entre el usuario y el algoritmo, de manera que pueda comprobar de forma sencilla e intuitiva el funcionamiento del algoritmo en diferentes entornos, así como la eficacia de los diferentes modelos definidos por el algoritmo.

Abstract

Video signal processing has undergo a constant evolution in the past few years. It may have happened because of its relevance in such a growing field as video security, therefore it has become so interesting the research in this area so that we innovate with new techniques or improve the existing ones.

In video signal processing, people detection is one of the most difficult tasks, even though there are actually some algorithms that give good results. However, when the detection takes part in complex environments the quality of the performance decreases, that is why the aim of this project is the implementation of a people detection in groups algorithm in C++, as well as the integration on a proprietary video analysis platform called DiVA. This display lets the execution with off-line videos, with the objective of verifying the functionality and efficiency in crowded environments.

The algorithm implemented, known as *Multi-configuration Part-based Person Detector,* is an adaptation of the algorithm known as DTDP (*Discriminatively Trained Part Based Models*) to improve the detection in these types of environments. It is founded in an exhaustive search of person models, which are formed by the mixture of different body parts. Thanks to this search it obtains good results despite of the processing time, therefore the analysis of this algorithm is not possible in real time.

In order to complete our goal, we have developed a user interface so that any user can interact with the algorithm and prove easily the functionality of the algorithm in different environments as well as test the efficiency of the different models defined.

Agradecimientos

Quiero agradecer, en primer lugar, a aquellos que han participado de forma activa en la realización de este trabajo: a mi tutor, Álvaro, por ayudarme a llegar hasta el final y asesorarme en todo momento, a José María Martínez y Jesús Bescos por haberme dado la oportunidad de realizar este trabajo en el grupo y a todos los compañeros del VPU, a Erik y María por conseguir que el laboratorio no siempre estuviera vacío y a Fulgencio, por su preocupación constante y su disposición para resolver mi eterna batería de dudas.

Gracias a todos los que han formado parte de estos años de universidad, aquellos que empezaron conmigo y los que se fueron incorporando, porque demostrasteis que lo importante en el viaje es el camino. A todos mis amigos a los que espero haber sido capaz de demostrar mi gratitud cada día, os dedico el mayor de los agradecimientos de este trabajo. A Sergio, por todo, pasado, presente y futuro.

No me puedo olvidar de las familias y amigos que formé en ese Río nuestro porque siempre estáis presentes, por lograr ese reencuentro. En especial Lucía, Erik, y Luis gracias por aparecer. Y a todos mis amigos de Burgos, GRACIAS, por todos los años que pasamos juntos y por conseguir que adore volver para poder veros.

Por último, le agradezco y se lo debo todo a mi familia. En especial a mis tíos, Pedro y María, por acogerme durante estos años como a una más; a Aurora y Blanca por estar en todo momento, por ser como sois y dejarme ser. A mis padres por haber luchado siempre por nosotras y por algún día devolveros todo aquello que nos disteis. Finalmente, a mi hermana por demostrar que no hace falta mucho para quererte. Ya sabes que es sencillo: contigo, siempre, hasta el final.

Índice general

Resumen v

Abstract vii

Agradecimientos ix

1. Introducción 1
1.1. Motivación . . . 1
1.2. Objetivos . . . 2
1.3. Estructura de la memoria . . . 3

2. Estado del arte 5
2.1. Introducción . . . 5
2.2. Algoritmo base . . . 7
2.2.1. Introducción . . . 7
2.2.2. Histogram of Oriented Gradient . . . 9
2.2.3. Discriminatively Trained Deformable Part-based . . . 10
2.2.4. Múltiples configuraciones de partes del cuerpo . . . 12
2.2.5. Determinación de nuevos umbrales . . . 14
2.3. DiVA . . . 15
2.4. Biblioteca QT . . . 16

3. Diseño 19
3.1. Latent SVM . . . 19
3.2. Múltiples configuraciones de partes del cuerpo . . . 22
3.3. Integración en DiVA . . . 26
3.3.1. Múltiples configuraciones de partes del cuerpo en DiVA . . . 27

4. Demostrador 29
4.1. Funcionalidad de la interfaz gráfica . . . 30
4.2. Caso de estudio . . . 33

5. Conclusiones y trabajo futuro 37

Índice de figuras

2.1. Arquitectura general de un detector de personas 6
2.2. Filtros de un detector por parejas 8
2.3. Ejemplo de descriptores HOG 9
2.4. Ejemplo de filtros en Latent SVM 10
2.5. Pirámide de características 11
2.6. Esquema general de DTDP 13
2.7. Ejemplos de las configuraciones 14
2.8. Arquitectura general de la plataforma DiVA 17

3.1. Esquema de la implementación de Latent SVM 21
3.2. Esquema de la implementación de múltiples configuraciones de partes del cuerpo. 22

4.1. Interfaz gráfica para el algoritmo de múltiples configuraciones 30
4.2. Ventana para la interfaz de selección de cámara. 31
4.3. Detalle de los botones de inicio y detención del algoritmo. 31
4.4. Detalle de la sección de parámetros. 32
4.5. Detalle de la ayuda de configuraciones 33
4.6. Resultado usando configuración básica 34
4.7. Resultado con las configuraciones 2, 7, 12 y 13. 34
4.8. Resultado con la configuración básica para otra secuencia de vídeo . . 35
4.9. Resultado empleando las diferentes configuraciones para otra secuencia de vídeo 35

Capítulo 1

Introducción

1.1. Motivación

En la actualidad existe una gran demanda en el área de la seguridad ésto, unido al gran avance de las tecnologías y en concreto del procesamiento de imagen y vídeo digital, hace de la vídeo-vigilancia una línea de investigación necesaria y que conviene desarrollar.

La vídeo-vigilancia se puede entender desde muchos puntos de vista y con muchas aplicaciones diferentes, como la detección de objetos, detección de caídas etc. la detección de personas es sólo una de estas aplicaciones, en la que se ha trabajado ampliamente, como prueba de ello tenemos la multitud de detectores de personas en visión por computador que existen en la actualidad. La complejidad de la detección de personas se encuentra, principalmente, en la dificultad para definir un modelo de las mismas, debido a la gran variabilidad en la apariencia física, poses, puntos de vista, movimiento e interacción entre las personas y los objetos e incluso entre otras personas.

Dentro de la detección de personas, la detección de personas en entornos de alta densidad sigue siendo un problema no resuelto. En este tipo de entornos la tarea de detección de personas se complica debido a que un alto porcentaje de las personas presentan algún tipo de oclusión. A la hora de analizar este tipo de escena se tendrá que tener en cuenta que, en general, no será posible tener una imagen de la persona entera.

Por todo lo expuesto, la motivación de este trabajo es lograr una implementación de un algoritmo, ya disponible en el estado del arte, en un nuevo lenguaje de programación, así como el desarrollo de una interfaz o demostrador, que facilite al usuario interaccionar con el detector y de esta forma comprobar de forma fácil, sencilla y

rápida que configuración, o conjunto de configuraciones, son las más apropiadas para cada escenario.

1.2. Objetivos

Una vez establecida la motivación de este trabajo, se plantean unos objetivos que abordar hasta alcanzar el objetivo final de manera progresiva. El desglose de los objetivos es el que sigue:

1. **Estudio del estado del arte**

   Análisis en profundidad de un método de detección de personas basado en modelos de diferentes partes del cuerpo como el **DTDP** ***(Discriminatively Trained Part Based Models)*** [1], donde cada parte del cuerpo es modelada según el algoritmo **HOG** ***(Histogram of Object Gradients)*** [2], así como su variación, el algoritmo base de este trabajo,[3], que permite el análisis de un entorno con alta densidad de personas de manera más o menos flexible.

2. **Aprendizaje de herramientas y bibliotecas**

   Utilizar las herramientas y bibliotecas necesarias para el desarrollo correcto de este trabajo: OpenCV [4] , DiVA *(Distributed Video Analysis Framework)* [5] y BibliotecaQt [6].

3. **Adaptación del algoritmo de detección de personas**

   Transformación e integración del código del algoritmo estudiado en el estado del arte utilizando el lenguaje C++ para que funcionen en dicho lenguaje y sobre la plataforma de análisis de vídeo denominada DiVA.

4. **Creación de interfaz gráfica**

   Desarrollo de una interfaz gráfica, utilizando la biblioteca Qt y el lenguaje de programación C++, a fin de que cualquier usuario pueda comprobar la funcionalidad del algoritmo.

El resultado final que buscamos es, por tanto, el desarrollo de un entorno sencillo y práctico para ejecutar el algoritmo presentado por [3], denominado *Multi-configuration Part-based Person Detector.*

1.3. Estructura de la memoria

La memoria del proyecto se divide en los siguientes capítulos:

- **Capítulo 1. Introducción:** motivación y objetivos del proyecto.
- **Capítulo 2. Estado del arte:** Descripción de tres algoritmos: HOG, DTDP y algoritmo base. Introducción a DiVA y biblioteca Qt.
- **Capítulo 3. Diseño:** Descripción de la adaptación y programación del algoritmo para su uso en la plataforma DiVA.
- **Capítulo 4. Demostrador:** Descripción de la funcionalidad de la interfaz gráfica.
- **Capítulo 5. Conclusiones y trabajo futuro:** Conclusiones, trabajo pendiente y trabajo futuro.
- **Referencias**

Capítulo 2

Estado del arte

2.1. Introducción

La detección de personas en secuencias de vídeo es una de las tareas más complejas incluidas dentro del tratamiento de señal de vídeo. Su complejidad reside principalmente en la dificultad que encontramos para definir un modelo genérico de las personas debido, en parte, a la gran variabilidad de las mismas: diferente apariencia, poses, movimientos o interacción con otras personas u objetos, etc.

Un factor importante en la detección de personas es conocer el entorno donde se desarrolla la acción ya que puede ayudar a desarrollar una mejor clasificación y puede ayudarnos a definir modelos más probables según la situación concreta con la que nos enfrentamos. Un ejemplo de la importancia del entorno es la diferencia entre modelos para entornos con pocas personas y entornos con mayor densidad de personas. En concreto este tipo de problema o escenario es el que desarrollamos en este trabajo.

Por este motivo, dividiremos el estado del arte en cuatro partes, primero trazaremos una pequeña introducción a la detección de personas, a continuación una explicación del estado del arte del algoritmo en el que basamos este trabajo, así como de los algoritmos que sirven como punto de partida para dicho algoritmo base. Finalmente daremos una breve explicación de la plataforma DiVA y de la biblioteca QT, herramientas que han sido necesarias para el desarrollo de este trabajo.

Detección de personas

Podemos dividir la detección de personas en tres etapas. En primer lugar se diseña y entrena un modelo de persona basado en las características de la misma pudiendo ser movimiento, silueta, dimensiones, etc. en segundo lugar se procede a una etapa de detección y por último una etapa de clasificación donde los objetos candidatos se

clasificarán como personas o no. En la figura 2.1 se puede ver la arquitectura básica de un detector de personas.

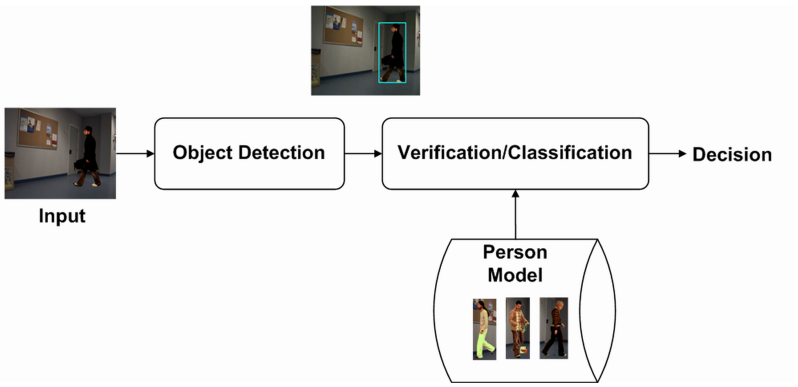


Figura 2.1: Arquitectura general de un detector de personas. Imagen extraída de [7]

Según los autores de [8] existen dos grandes grupos dentro de la clasificación de objetos: aquellos que buscan coincidencias en la silueta y los que buscan coincidencias en la apariencia. Para algoritmos basados en la silueta encontramos un ejemplo simple en [9] donde se buscan correspondencias entre la forma de la parte superior de las personas y un modelo mediante correlación tras una segmentación basada en simetría. Otros ejemplos como [10, 11] realizan la comparación con un enfoque algo más sofisticado y proponen un algoritmo jerárquico de correspondencia con las siluetas, partiendo de un ajuste más grueso hasta llegar a uno más fino. Aunque estos modelos son más simples y fáciles de implementar que las técnicas basadas en la apariencia también resultan menos precisos al no soportar variaciones de postura.

Por otra parte, aquellas técnicas que se basan en la apariencia definen imágenes de características (descriptores) a partir de las cuales se clasifican las ROI (*Region of Interest*) que puedan contener personas y se descartan aquellas que no las contengan. Como ejemplo de este tipo de algoritmo en [2] se presenta *Histograms of Oriented Gradient* (HOG) donde la idea básica es que la apariencia y forma de los objetos se puede caracterizar tanto mediante la distribución de los gradientes de intensidad local como por las direcciones de los bordes. Aunque HOG consigue buenos resultados su coste computacional es alto, como explicaremos detalladamente en la sección 2.2.2.

Detección de personas en grupos

La detección de personas en zonas muy concurridas es un problema añadido que ha de tenerse en cuenta a la hora de plantear cualquier algoritmo de detección de personas. Existen algoritmos de detección como [1] que comienzan a fallar con oclu-

siones de un 20 % entre objetos y que por encima de este porcentaje las detecciones se vuelven cada vez más inusuales. Por este motivo se desarrollan otras herramientas para detectar personas en grupos, como por ejemplo los métodos [12, 13] incluyen herramientas de *tracking* para seguir a personas que son ocluidas durante períodos largos de tiempo. Este y otros métodos pensados para grupos sólo basan su búsqueda en la información de las partes de las personas que siguen siendo visibles pero existen otros métodos como [14] en los que los autores aprovechan la idea de que en escenas con multitudes de personas, las oclusiones son producidas por unas personas sobre otras en su gran mayoría y, por tanto, se puede aprovechar esta información para desarrollar patrones de características que se crean al solaparse personas entre sí. Los autores desarrollan un modelo de detector por parejas consistente en un filtro *root* que define la localización de las dos personas y n filtros de partes deformables que cubren las partes más características de cada persona así como patrones de oclusión de las dos personas. Una vez detectada una pareja, el sistema calculará la posición de las cajas de detección de cada persona individual a través de un modelo de regresión lineal a fin de identificar a cada una de las personas que forma la pareja. En la figura 2.2 podemos ver un ejemplo de los filtros usados la detección de parejas.

Dentro de la detección de personas en entornos de mayor densidad hemos dedicado este trabajo al algoritmo presentado en [3], que adapta el sistema de detección desarrollado por [1], conocido como DTDP y que basa la detección en mezclas de modelos de partes deformables, para conseguir una mejor detección de personas en grupos. Explicaremos ambos algoritmos de forma más detallada a continuación, en la sección 2.2.

2.2. Algoritmo base

2.2.1. Introducción

El reconocimiento de objetos es uno de los mayores retos del tratamiento de imágenes por ordenador, esto se debe a que, incluso dentro de una misma categoría, los objetos tienen una gran variabilidad no sólo debida a cambios en la iluminación, el color o ángulos de visión sino también debida a la propia variación de la forma. En el caso concreto de la detección de personas podemos observar diferencia en ropas, colores, posturas, tamaños, etc. que confieren una gran variabilidad. Dado un objeto, éste puede ser dividido en partes y, cada una de ellas, tendrá ciertas propiedades comunes a esa región, mientras que la componente deformable puede ser caracterizada por la conexión entre pares de partes cercanas, por eso existen algoritmos que utilizan diferentes configuraciones de partes de las personas, basadas en la fisionomía de estas, ya

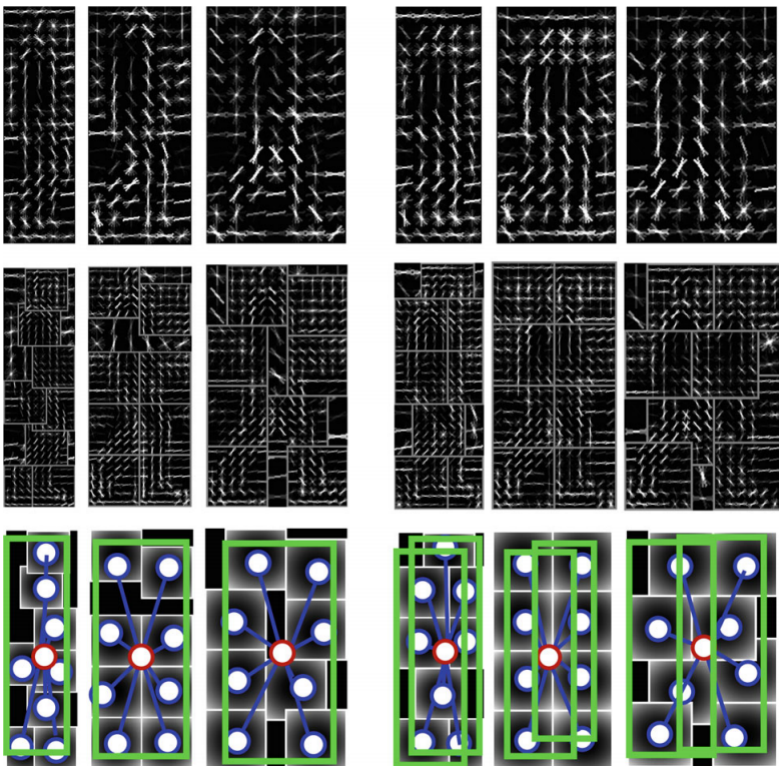


Figura 2.2: Visualización de filtros *root*, filtros por partes y cajas de detección de un modelo por parejas. Las tres primeras columnas corresponden a filtros de una persona y las tres últimas a filtros por parejas. Imagen extraída de [14]

sean definidas como un conjunto, como explicaremos en la sección 2.2.3 o haciendo uso de sólo algunas de ellas [3, 15] como cabeza, hombros, piernas, etc. o combinaciones de las mismas. Se ha de tener en cuenta que, en las situaciones más comunes dentro de la detección de personas en grupos, las extremidades inferiores, normalmente, no serán visibles o estarán ocluidas por otras partes superiores de otras personas y, por ello, se espera que gracias a la elección de las diferentes combinaciones de partes del cuerpo aumente el rendimiento del sistema. En algunas publicaciones como [15] se aprovecha la combinación de diferentes partes del cuerpo y se permite que ellas no puntúen en el centro de la persona sino en cualquier otro punto que convenga.

Este tipo de detección es la que desarrollaremos y explicaremos en las secciones 2.2.4 y 2.2.5. Antes, en las secciones 2.2.2 y 2.2.3, explicaremos brevemente el estado del arte de dos algoritmos que forman la base del algoritmo en el que basamos este trabajo y que, por tanto, facilitarán el entendimiento del mismo.

2.2.2. Histogram of Oriented Gradient

Este método propuesto por [2] consiste en la evaluación de histogramas locales normalizados de las orientaciones de los gradientes de una imagen. Se implementa dividiendo la imagen en pequeñas regiones espaciales (*cells*), se acumula en cada una de ellas un histograma, de una dimensión, de las direcciones de gradiente o de las orientaciones del borde sobre los píxeles que se encuentran dentro de cada *cell*. La combinación de estos histogramas forma la representación que vemos en la imagen 2.3.

(a) (b) (c)

Figura 2.3: Ejemplo de descriptores HOG. (a) Promedio de los gradientes en la imagen de entrenamiento. (b) Imagen de entrenamiento. (c) Descriptor HOG. Imagen extraída de [2]

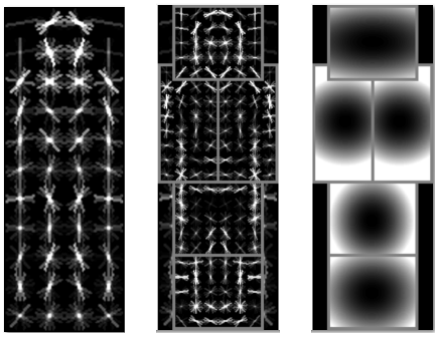
Para conseguir una mejor invariancia frente a la iluminación, las sombras, etc., se recomienda normalizar el contraste de las zonas locales antes de operar con ellas. Esto se puede conseguir acumulando la energía de los histogramas locales sobre una región mayor (*blocks*) y usar este resultado para normalizar todos los *cells* que están contenidos en dicho *block*. A estos bloques normalizados los llamaremos descriptores HOG (*Histogram of Oriented Gradient*). Estos descriptores se obtendrán de una ventana de detección dividida en una cuadrícula, con cierto grado de solape entre cuadros y usando SVM (*Support Vector Machine*) lineal se hará la clasificación como personas o no.

HOG captura los bordes o estructuras de gradientes más característicos de una zona local y lo hace con un cierto grado de invariancia a transformaciones, es decir, las traslaciones o rotaciones apenas afectan al resultado.

Como ya comentamos antes, HOG es un método que obtiene buenos resultados pero tiene un alto coste computacional que lo convierte en un algoritmo lento.

2.2.3. Discriminatively Trained Deformable Part-based

[1] describe un sistema de detección basado en mezclas de modelos de partes deformables multiescala, *Discriminatively Trained Deformable Part-based model* (DTDP), definido por un filtro *root*[1] que, junto con el resto de partes deformables del cuerpo, están modelados por HOG, como en primer lugar propuso [2]. El detector propone N partes del cuerpo posicionadas alrededor del *root* $(n = 0)$ como se muestra en la imagen 2.4. Estos filtros correspondientes a las diferentes partes del cuerpo están definidos al doble de resolución y cubren pequeñas zonas del cuerpo, tal y como puede verse en la figura 2.5, estos filtros de pequeñas partes del cuerpo identifican cada zona en mayor detalle.



(a) (b) (c)

Figura 2.4: Ejemplo de filtro *root* y filtros de partes del cuerpo en Latent SVM. Donde (a) corresponde con el filtro *root,* (b) con los filtros de cada parte del cuerpo al doble de resolución y (c) con el modelo espacial de la localización de cada filtro con respecto al *root.* Imagen extraída de [1]

Cada parte del modelo, incluyendo el *root,* $(n = 0, ..., N)$ está definido por tres variables F_n, $v_{n,0}$, d_n; donde F_n es la respuesta al filtro HOG para la parte n. Por su parte, $v_{n,0}$ es un vector de dos dimensiones que define la posición relativa de la parte n, con respecto al *root*, y d_n es un vector de cuatro dimensiones que define los coeficientes de una función cuadrática que define la deformación de cada parte n. BP_n representa la puntuación de un píxel en la posición (x, y) para la parte del cuerpo n donde s identifica la escala $(s = 1, ..., S)$. Por tanto, la puntuación de cada parte, se obtiene de las siguientes ecuaciones 2.1, 2.2 y 2.3.

$$BP_n(x, y, s) = F_n(x, y, s) - \langle d_n, \phi(dx_n, dy_n) \rangle \tag{2.1}$$

[1] Filtro principal que define el objeto en su totalidad.

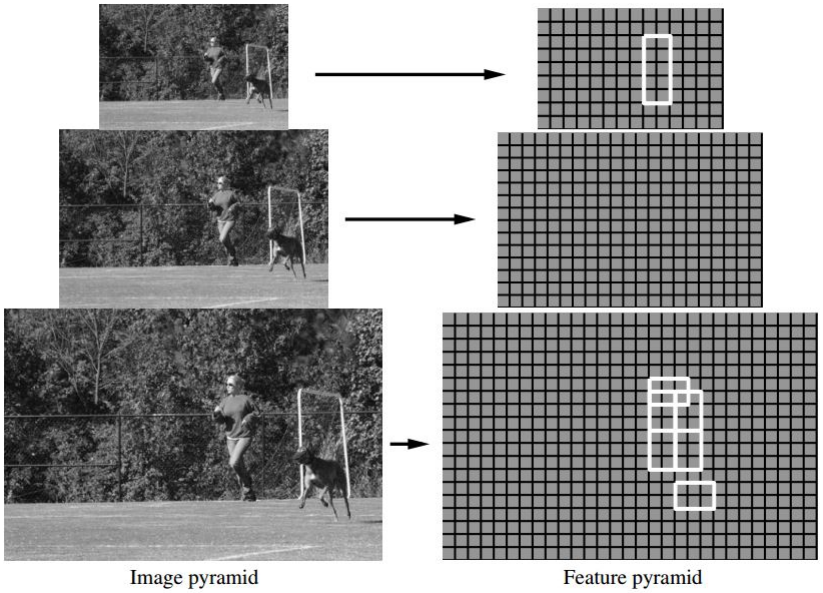


Figura 2.5: Pirámide de características. Los filtros de partes están en el nivel de la pirámide que les confiere doble resolución que el filtro root. Imagen extraída de [1].

$$(dx_n, dy_n) = (x_n, y_n) - (2\,(x_0, y_0) + v_{n,0}) \tag{2.2}$$

$$\phi\,(dx, dy) = \left(dx, dy, dx^2, dy^2\right) \tag{2.3}$$

Donde 2.2 representa el desplazamiento de la parte n con respecto al *root* y 2.3 la distribución de la deformación espacial de la parte n.

El resultado final de la detección viene dado por $C(x, y, s)$, como la suma del *root* y todas las partes en cada píxel y escala. Una puntuación alta determinará una detección, aunque como método adicional para evitar falsos positivos se eliminan aquellas detecciones que tengan un solape superior al 50 %. El umbral escogido, por encima del cual se detectará una persona, depende en [1] directamente del número total de partes del cuerpo detectadas.

$$C\,(x, y, s) = \sum_{n=0}^{N} BP_n\,(x, y, s) \tag{2.4}$$

A modo de ejemplo en la figura 2.6 podemos ver un esquema de como funciona DTDP. Se crean dos mapas de características uno de ellos se comparará con el filtro *root,* mientras que el otro, al doble de resolución, se comparará con los filtros por

partes. De la combinación de todas las puntuaciones obtenemos el resultado final y la detección.

La principal dificultad de este tipo de sistemas se encuentra en que son muy complicados de entrenar debido a que en muchas ocasiones se necesita información latente[2], las imágenes de entrenamiento sólo disponen de la información de un rectángulo alrededor del objeto, se desconoce, por tanto, dónde están situadas las diferentes partes del cuerpo. Con un etiquetado se podría aportar dicha información, aunque supondría un importante gasto de tiempo.

2.2.4. Múltiples configuraciones de partes del cuerpo

Partiendo de la base de DTDP explicada en la sección 2.2.3, en [3] se propone un método para mejorar este algoritmo aplicado a la detección de personas en grupos permitiendo que el algoritmo no base su decisión únicamente en una sola configuración de las partes del cuerpo detectadas. Para ello propone el uso de diferentes configuraciones de partes del cuerpo de personas, t $(t = 1, ..., T)$ donde $1 \leq T \leq 2^N$ y cada configuración t se compone de un subconjunto de las N partes del modelo original, [1]. Este algoritmo diseña treinta configuraciones diferentes $(T = 30)$. Estas configuraciones se diseñan pensando en la aplicación específica de detección de personas en grupos por lo que se ha tenido en cuenta el tipo de oclusión que se da en estos casos, siendo en muchos casos generadas por otras personas e imposibilitando ver las extremidades inferiores, por eso se diseñan configuraciones donde sólo se tienen en cuenta partes potencialmente visibles del cuerpo. Se diseñan quince configuraciones, combinando diferentes partes del cuerpo, y otras quince configuraciones, idénticas a las anteriores, que contendrán, además, el filtro *root*. Todas las configuraciones van añadiendo progresivamente partes del cuerpo desde la cabeza $(t = 1)$, hasta todas las ocho partes del cuerpo definidas $(t = 15)$. En la figura 2.7 se puede ver el ejemplo de alguna de estas configuraciones.

[2] Información que no puede ser observada a simple vista

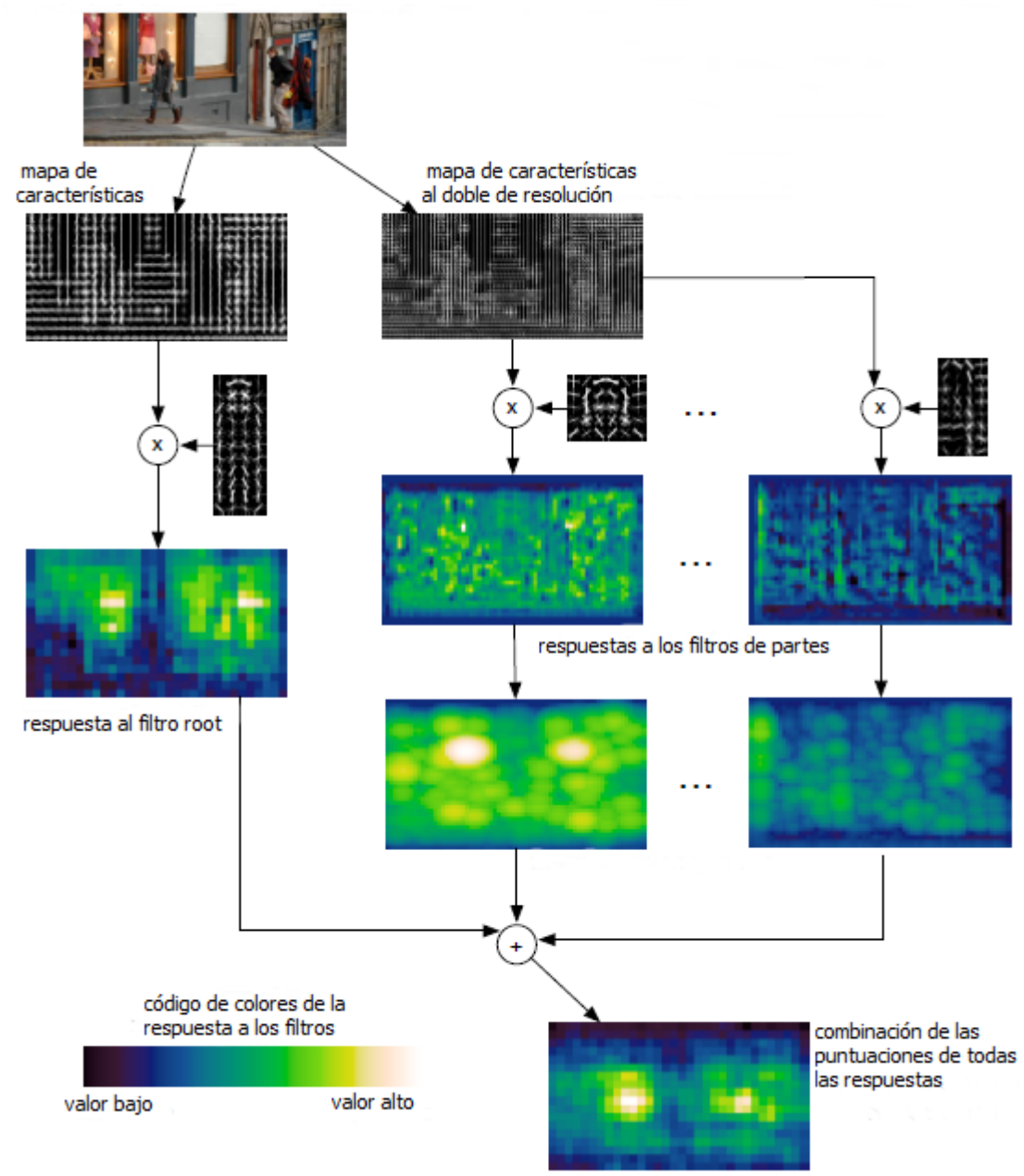


Figura 2.6: Esquema general de DTDP. Imagen extraída de [1]

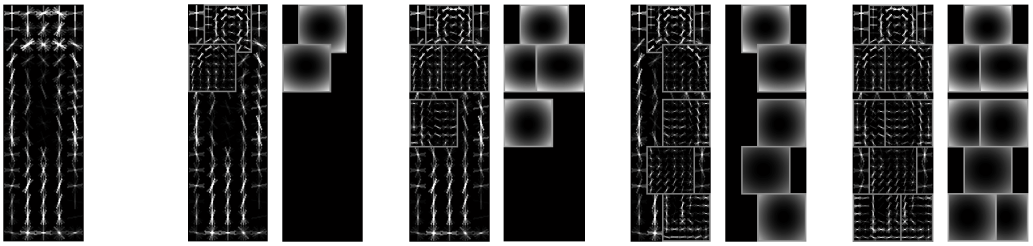


Figura 2.7: Ejemplos de las configuraciones. La primera columna se corresponde con el filtro *root*. El resto de columnas corresponde a las configuraciones diseñadas, a cada configuración se le ha dado un código numérico.

Por último, a diferencia del algoritmo original (ver sección 2.2.3, ecuación 2.4) el resultado para cada configuración es definido por 2.5 y 2.6 donde α^t es un vector binario para cada configuración.

$$C_t\left(x,y,s\right) = \sum_{n=0}^{N} \alpha_n^t \cdot BP_n\left(x,y,s\right) \tag{2.5}$$

$$\alpha_n^t = \begin{cases} 1, & n \subset t \\ 0, & otherwise \end{cases} \tag{2.6}$$

Para este caso particular es necesario determinar un umbral diferente para cada configuración que dependerá del número y tipo de las partes del cuerpo que formen dicha configuración, tal y como explicaremos a continuación.

2.2.5. Determinación de nuevos umbrales

Como ya hemos comentado, para el algoritmo presentado por [3], es necesario determinar un umbral para cada configuración, que dependerá del número y tipo de partes que forman cada configuración, ya que no todas las partes tienen el mismo peso para las diferentes configuraciones. Para ello, si se considera BP_n como el resultado de cada parte del cuerpo y su distribución de densidad de probabilidad $f_{BP_n}\left(bp_n\right)$, el umbral final para cada configuración será una variable aleatoria continua C_t y su distribución de densidad de probabilidad asociada $f_{Ct}(c_t)$. Para estimar el mínimo umbral k_t que se necesitará para considerar un objeto detectado como persona, para cada configuración, [3] propone determinar un factor R_t que tenga en cuenta el número de partes del cuerpo que forman la configuración y su relevancia en relación a la configuración original de N partes. Por ejemplo, asumiendo que cada parte contribuye de igual forma al modelo de persona este factor podría ser $R_t = \frac{1}{N}$ para cada

configuración t. Para estimar la contribución de cada parte del cuerpo n, se estima la relación entre la distribución de puntuaciones obtenidas usando todas las partes del cuerpo F_C y la distribución de puntuaciones obtenidas usando todas las partes del cuerpo menos la parte en estudio. El factor de corrección del umbral para cada configuración vendrá dado por R_t, que se calcula como la suma de la contribución de cada parte del cuerpo, tal como se expresa en 2.7, donde $K\bar{L}_n$ expresa la contribución de cada parte del cuerpo, calculada con la divergencia de Kullback-Leibler [16], normalizada de modo que $\sum_{n=1}^{N} K\bar{L}_n = 1$.

$$R_t = \sum_{n=0}^{N} \alpha_n^t \cdot K\bar{L}_n \tag{2.7}$$

Esto quiere decir que una configuración con factor de corrección $R = 1$ no necesitará cambiar el umbral con respecto al umbral original porque dicha configuración cuenta con todas las partes del cuerpo. Todos los objetos evaluados que superen este umbral quedarán clasificadas como personas.

Por último, el método propuesto por [3], al igual que en DTDP, sigue eliminando aquellas detecciones que superen un cierto nivel de solape, ahora este proceso también se aplica a los resultados obtenidos con todas las configuraciones de modelos de personas juntas.

2.3. DiVA

El objetivo de este trabajo es la implementación del algoritmo explicado en la sección 2.2.4 en C++ así como su inclusión en la plataforma DiVA [5] desarrollada por el *Video Processing and Understanding Lab (VPU)*.

La plataforma DiVA (*Distributed Video Analysis Framework*) [5] establece un **entorno distribuido** para la intercomunicación simultánea de múltiples fuentes de vídeo con algoritmos de proceso, la conexión en cascada de estos algoritmos de proceso, la visualización de resultados parciales, y la inclusión formalizada de información contextual en el proceso de análisis, todo ello posibilitando que los flujos de datos se procesen en tiempo real.

Tanto la plataforma como las aplicaciones que hacen uso de ella han sido desarrolladas e implementadas por el *Video Processing and Understanding Lab* (VPU) de la Escuela Politécnica Superior de la Universidad Autónoma de Madrid.

Este sistema es escalable (puede añadir módulos de procesado adicionales), eficiente (opera sin apenas incrementar el coste computacional total), generalizable (sus funcionalidades se desarrollan usando herramientas y protocolos genéricos) y con to-

lerancia a fallos.

Se propone un desarrollo modular en distintos subsistemas. Cada subsistema está relacionado con una función específica dentro de la plataforma. Estas funciones contemplan desde la captura de datos desde distintos dispositivos físicos hasta el almacenaje y presentación de los resultados obtenidos por los algoritmos, todo ello de una forma distribuida. Los subsistemas que componen la plataforma se muestran en la Figura 2.8.

- El **subsistema de captura de datos** reserva los recursos necesarios para su funcionamiento y comienza a capturar datos de las fuentes de vídeo que han sido seleccionadas para capturar la señal de vídeo.
- El **subsistema de bases de datos** proporciona un contexto de aplicación a los algoritmos de análisis y almacena los resultados del procesado de otros módulos de análisis.
- El **subsistema de procesamiento** realiza el propio procesado de la señal de vídeo y proporciona una interfaz de trabajo para cualquier algoritmo de procesado de vídeo.
- El **subsistema de presentación de datos** presenta en pantalla los datos de los análisis resultantes del subsistema de procesamiento.

La plataforma está desarrollada bajo un modelo cliente/servidor en el que se separa la parte servidora de contenido (subsistemas de captura y almacenamiento de datos) de la parte que lo consume (subsistemas de procesamiento y presentación de datos).

2.4. Biblioteca QT

Además de implementar el algoritmo explicado en la sección 2.2.4 en C++, este trabajo busca crear una interfaz para que cualquier usuario pueda ver los resultados del algoritmo con los parámetros por defecto así como modificar ciertos parámetros, para ello hemos utilizado la biblioteca Qt.

Qt [6] es una biblioteca multiplataforma usada para el desarrollo de aplicaciones de interfaz gráfica, aunque también se puede usar para desarrollar programas sin interfaz gráfica. Qt es un software libre y de código abierto desarrollado por Qt Project, donde participan tanto miembros de su comunidad como empresas del sector tecnológico.

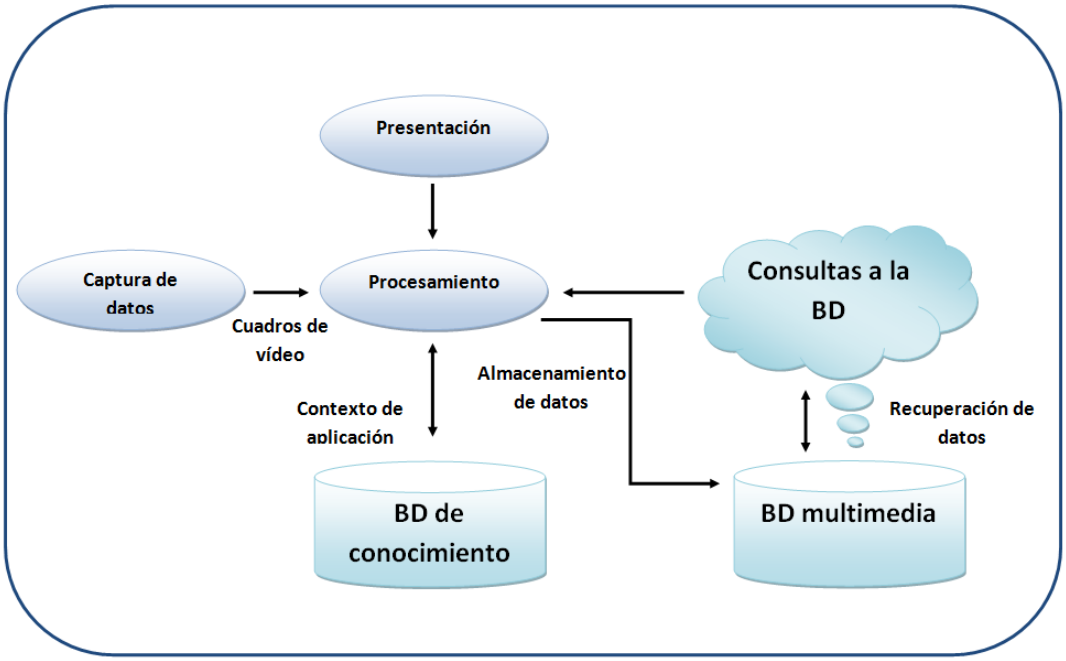


Figura 2.8: Arquitectura general de la plataforma DiVA

Qt utiliza el lenguaje de programación C++, aunque puede ampliarse a otros lenguajes a través de *bindings*, es decir, adaptaciones de bibliotecas para que puedan ser usadas en otros lenguajes de programación.

En este trabajo hemos hecho uso de la biblioteca Qt para desarrollar la interfaz gráfica del algoritmo de detección de personas explicado en la sección 2.2.4 y 2.2.5. Dicha interfaz se encuentra explicada en detalle en el capítulo 4.

Capítulo 3

Diseño

En este capítulo describiremos en detalle la implementación en C++ e integración en la plataforma DiVA del algoritmo explicado en la sección 2.2 del estado del arte. Para ello hemos desarrollado tres etapas, en este capítulo sólo explicaremos las dos primeras, dejaremos el desarrollo de la interfaz gráfica para el siguiente capítulo:

1. **Adaptación del algoritmo a C++:** se ha creado una clase C++ siguiendo unas pautas de desarrollo tomando como partida uno de los algoritmos estudiados en el estado del arte.

2. **Encapsulación del algoritmo** a partir del encapsulador DiVAAlgorithm presente en la plataforma.

3. **Desarrollo de la interfaz gráfica**. Esta etapa se desarrollará en el capítulo 4.

3.1. Latent SVM

Para poder describir la adaptación del algoritmo disponible en Matlab [3], explicado en la sección 2.2, en primer lugar vamos a describir la versión disponible en OpenCV del detector base original en el que se basa nuestro algoritmo, descrito en la sección 2.2.3 del estado del arte y conocido como **DTDP** [1], dicho algoritmo se encuentra disponible en OpenCV bajo el nombre de **Latent SVM,** basándonos en esta implementación hemos implementado nuestro algoritmo. Por este motivo en primer lugar describiremos Latent SVM.

En la figura 3.1 se puede ver de forma simplificada el esquema que sigue Latent SVM y se encuentra implementado de la siguiente manera:

- Primero un constructor carga el **modelo de persona.** Para ello se crea un objeto, *detector*, perteneciente a la clase *cvLatentSvmDetector,* sobre el que se

ejecutará la función *cvLoadLatentSvmDetector,* que cargará el modelo entrenado desde el archivo indicado.

- Una vez que tenemos el modelo, el **procesador** se encarga de analizar cada uno de los *frames* recibidos con el fin de detectar personas en ellos. Para ello se hará uso de la función *cvLatentSvmDetectObject,* que encontrará zonas rectangulares en la imagen que tengan niveles de confianza adecuados como para ser considerados personas. Para ello llamamos a la función de la siguiente forma:

  CvSeq cvLatentSvmDetectObjects(IplImage* image,CvLatentSvmDetector* detector,CvMemStorage* storage, float overlap_threshold=0.5f, int numThreads=-1)*

  Donde:

  - ***Image*** corresponde al frame que se está procesando.
  - ***Detector*** contiene el modelo de personas entrenado.
  - ***Storage*** especifica un almacenamiento en memoria donde se guardarán la secuencia resultante de rectángulos encontrados y puntuaciones correspondientes.
  - ***Overlap_threshold*** contiene el umbral permitido de solape entre detecciones.
  - ***NumThreads*** especifica el número de hilos permitido, para aquellas ejecuciones que permitan una ejecución con varios hilos.

  Dentro de *cvLatentSvmDetectObjects* la detección se hará en tres pasos principales:

  - En primer lugar se crea una **pirámide de características** usando *createFeaturePyramidWithBorder* que calcula mapas de características de la imagen a diferentes resoluciones. Generará múltiples niveles o resoluciones en función del tamaño de la imagen.
  - En segundo lugar obtiene las **detecciones** con la función *searchObjectThresholdSomeComponents,* que recibe como argumentos la pirámide de características, los filtros HOG [2], tanto el *root*, filtro principal que define en su totalidad el objeto, como los filtros que definen las partes del cuerpo al doble de resolución, tal y como se mostraba en el ejemplo de la figura 2.6, en la sección 2.2.3. Esta función también recibe como argumento el umbral de detección con el que comparará las puntuaciones, es decir, cualquier objeto detectado con una puntuación por encima de este umbral será

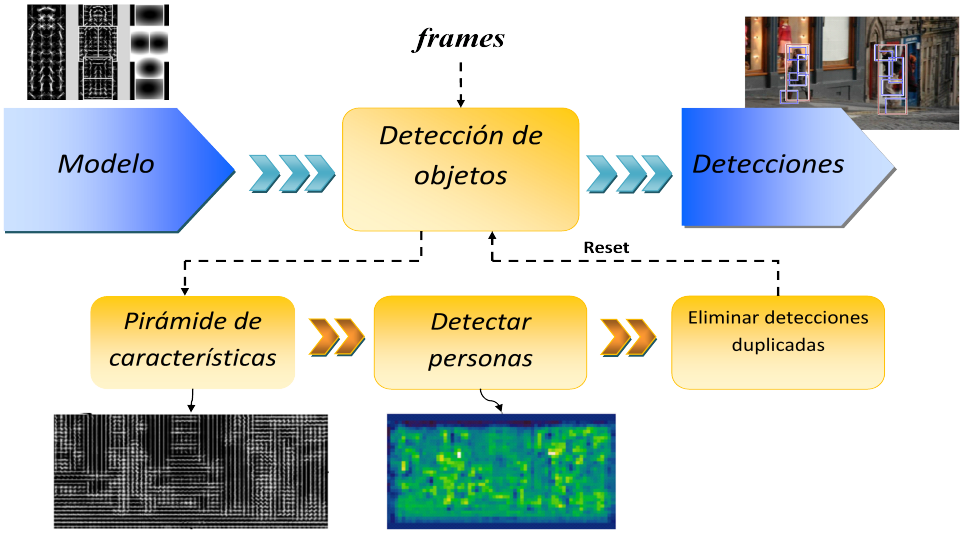


Figura 3.1: Esquema de la implementación de Latent SVM

clasificado como persona. Una vez realizadas las detecciones esta función devuelve el desplazamiento de los filtros, un vector con las puntuaciones y el número de detecciones encontradas.

- ✦ Por último, es necesario eliminar posibles detecciones duplicadas para evitar **falsos positivos** mediante *nonMaximumSuppression* que eliminará aquellas detecciones que superen un cierto nivel de solape.

- Finalmente, una vez que se tienen todas las detecciones se dibujan mediante la función *cvRectangle,* que dibujará un rectángulo por cada detección.

Debido a lo exhaustivo de Latent SVM no se puede analizar este algoritmo en tiempo real, por ello ejecutamos tanto ésta como nuestra implementación con vídeos pertenecientes a una base de datos aunque sería posible ejecutar sobre una cámara procesando con un frame rate muy bajo.

Una vez que conocemos la implementación de Latent SVM en OpenCV explicaremos la implementación del algoritmo en el cual se basa este trabajo y su integración en la plataforma DiVA, para ello explicaremos la implementación llevada a cabo para el algoritmo de múltiples configuraciones, sección 3.3.1 y daremos una breve explicación de los pasos dados para integrar el algoritmo en la plataforma DiVA, secciones 3.3 y 3.3.1.

3.2. Múltiples configuraciones de partes del cuerpo

Para implementar el algoritmo presentado en [3] nos hemos basado en la implementación básica de Latent SVM desarrollada en OpenCV (ver sección 3.1) a fin de que, a partir de ésta, podamos añadir la capacidad de detectar personas usando varios tipos de configuraciones diferentes, siguiendo lo ya explicado en la sección 2.2.4 del capítulo de estado del arte, y permitiendo que sea el usuario el que elija entre las diferentes configuraciones para detectar personas así como la decisión del umbral y el solape.

Para ello hemos creado una clase derivada de *cvLatentSvmDetector* a la que hemos nombrado ***cLatentSvmDetector***, dentro de esta clase hemos declarado nuevos métodos y constructores con las modificaciones necesarias para implementar el algoritmo. El algoritmo implementado, [3], sigue un esquema muy parecido al de Latent SVM. Sin embargo, tiene algunas diferencias, como se puede apreciar en el esquema simplificado que se muestra en la figura 3.2.

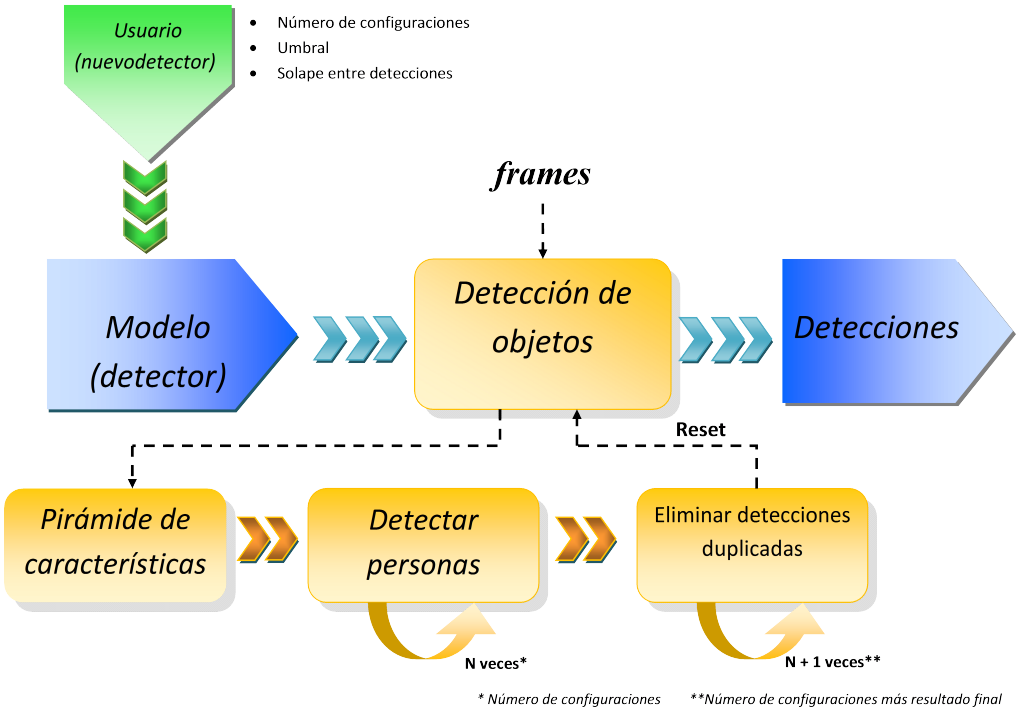


Figura 3.2: Esquema de la implementación de múltiples configuraciones de partes del cuerpo.

- **Constructores y destructores**

  Los constructores de una clase son necesarios para inicializar variables, así como reservar la memoria necesaria para el correcto funcionamiento del algoritmo. Además de los constructores ya implementados en OpenCV hemos implementado dos nuevos constructores que servirán para leer y escribir un fichero de configuraciones que contendrá la información de las diferentes configuraciones.

  El primero de estos constructores lo usaremos para inicializar un objeto, al que llamamos ***nuevodetector***, de la siguiente forma:

  *ml::CLatentSvmDetector *nuevodetector= new ml::CLatentSvmDetector (this->model_filename, this->config, this->file_config);*

  Nuevodetector se encargará de escribir un nuevo fichero, *model_filename,* que tendrá el mismo formato que el fichero de configuraciones original, *file_config,* y que se irá actualizando añadiendo o eliminando las configuraciones que elija el usuario. Hemos decidido iniciar el algoritmo por defecto con todas las configuraciones posibles, en este caso 30, por lo que, en este punto, *model_filename*[1] debe ser idéntico a *file_config*[2]. Para crear el nuevo fichero de configuración hemos desarrollado un método, ***write_configurations***, que pertenece a *CLatentSvmDetector,* que creará y escribirá un nuevo fichero de configuración a partir de la información del fichero original.

  El segundo de estos constructores servirá para inicializar otro objeto, al que llamamos ***detector***, de la siguiente forma:

  *ml::CLatentSvmDetector *detector= new ml::CLatentSvmDetector (this->model_filename,this->file_config);*

  Detector se encargará de leer los datos del fichero de configuraciones generado por el usuario que contiene los datos necesarios para la detección, con estos datos inicializará cada variable con el valor correspondiente. Para leer el fichero hemos desarrollado un método ***parse_configurations***, perteneciente a *CLatentSvmDetector,* este método, a parte de leer el fichero de configuración, calcula los niveles de confianza, es decir umbrales, para cada configuración que, como

[1] Fichero de configuración generado por el usuario
[2] Fichero de configuración original que sirve de punto de partida para generar *model_filename*

explicamos en la sección 2.2.5 del estado del arte, será diferente para cada configuración en función del número y tipo de partes del cuerpo que compongan la configuración.

Por su parte, el destructor declarado como *~CLatentSvmDetector()* libera la memoria reservada para todas las configuraciones.

- **Métodos**

  Para la implementación del algoritmo en C++ ha sido necesario modificar algunos de los métodos desarrollados por OpenCV, a fin de aumentar su funcionalidad, así como implementar nuevos métodos, como los explicados en el apartado anterior, *parse_configurations* y *write_configurations,* y otros, que explicaremos a continuación, necesarios para permitir al usuario escoger entre las posibles configuraciones y modificar ciertos parámetros.

  En primer lugar para detectar las personas hemos adaptado la función ***cvLatentSvmDetectObjects***, el número de argumentos de nuestra función varía con respecto a la original y queda declarada de la siguiente forma:

  *CvSeq** cvLatentSvmDetectObjects(IplImage* image,CvLatentSvmDetector* detector, CvMemStorage* storage, float *overlap_threshold, int numThreads, float *score_threshold, int N_configurations, int **configuration)*

  A esta nueva función le pasamos los umbrales de cada configuración, *score_threshold*, así como el número de configuraciones, *N_configurations,* y los datos de todas las configuraciones, *configuration.* Dentro de esta función ha sido necesario la adaptación de ciertas funciones con respecto a la versión de OpenCV:

  - Al igual que en Latent SVM esta función llamará a ***createFeaturePyramidWithBorder*** para calcular el mapa de características de la imagen. Esta función no ha sido necesaria modificar, ya que el mapa se genera con respecto a la imagen y, por tanto, será el mismo para todas las configuraciones.
  - Sin embargo, ***searchObjectThresholdSomeComponents*** sí que ha sido necesaria adaptarla para detectar personas usando todas las configuraciones. Esta función se encargará de comparar la imagen con todos los filtros y devolver un array de las posiciones de los filtros, tanto del *root* como de las diferentes partes que forman el modelo. Nuestra implementación deberá detectar personas para una imagen buscando coincidencias con todos los modelos que desee el usuario, es decir, deberá comparar una misma imagen

con todas las diferentes configuraciones escogidas por el usuario. Para ello debemos modificar la reserva de memoria para que pueda guardar los datos de todas las detecciones de manera correcta y deberá comparar el mapa de características, tantas veces como sea necesario, con los diferentes modelos, es decir, ahora deberá comparar para cada mapa de características a cada resolución los diferentes filtros que componen cada configuración. Todos aquellos objetos que obtengan una puntuación mayor al umbral serán declarados como personas, por eso, esta función por tanto devolverá las coordenadas y puntuaciones de las detecciones obtenidas con las diferentes configuraciones.

Una vez obtenidos estos arrays de puntos y puntuaciones dentro de *cvLatentSvmDetectObjects* se deberán eliminar las posibles detecciones duplicadas, al igual que en Latent SVM, teniendo en cuenta que ahora se deberá aplicar este proceso tanto a las detecciones de las diferentes configuraciones como a la detección final, resultado de combinar todas las detecciones.

Paralelamente se han desarrollado, o modificado con respecto a la versión de LatentSVM [4], funciones auxiliares para llevar a cabo la detección, la eliminación de posibles detecciones duplicadas, la estimación de las cajas de detección, etc. algunas de estas funciones son ***searchObjectThreshold, nonMaximunSuppresion, clippingboxes, estimateBoxes, etc****.*

Finalmente dibujamos, haciendo uso de ***cvRectangle****,* las detecciones resultantes.

Método para modificar los parámetros:

Por último, para lograr un algoritmo con el que un usuario pueda interactuar hemos implementado el algoritmo de modo que se permita al usuario escoger las configuraciones con las que se quiere detectar personas, así como escoger el umbral de detección base, es decir, el del algoritmo o configuración original y el solape entre detecciones. Para ello se han desarrollado dos métodos que permiten sobrescribir el fichero de configuraciones y actualizar el objeto declarado como *detector* para que modifique los parámetros durante la ejecución.

Estos métodos son miembros de *CLatentSvmDetector* y están declaradas de la siguiente forma:

virtual bool rewrite(const std::vector<std::string>& filenames, int config[30], char file_configurations=NULL, const std::vector<std::string>& classNames=std::vector<std::string>());*

virtual bool update(const std::vector<std::string>& filenames, char file_configurations=NULL, const std::vector<std::string>& classNames=std::vector<std::string>());*

Donde *rewrite* recibe un vector que contiene una lista de las configuraciones seleccionadas por el usuario. A partir de la información contenida en el fichero original sobrescribirá el fichero del usuario para que contenga sólo las nuevas configuraciones elegidas. A su vez *update* leerá el nuevo fichero generado y actualizará los valores contenidos en *detector*.

Cuando el usuario modifique el umbral o el solape será necesario reasignar los valores de las variables correspondiente de la siguiente forma:

this->detector.thr_original = valor_float;

this->detector.overlap_original = valor_float;

donde *valor_float* es el nuevo valor seleccionado por el usuario y después actualizar esta información con *update*.

3.3. Integración en DiVA

Con el fin de integrar nuestro algoritmo creamos una clase C++ que facilitará la integración en DiVA y que contará con las siguientes características:

- Un **constructor** que inicialice todos los parámetros, o los lea de un fichero de configuración, con valores por defecto y reserve los recursos necesarios para la ejecución.
- Un **destructor** que se encargue de liberar los recursos necesarios.
- Un **método público** encargado de procesar cada *frame* y que tendrá la siguiente sintaxis:

  *void * process (IplImage *frame)*

  Recibirá una imagen en formato OpenCV y se encargará de procesarla.

- Un **método público** encargado de mostrar los resultados con el siguiente formato:

  void showresults();

- Métodos **set/get** que servirán para configurar los parámetros.

DiVAAlgorithm es una clase desarrollada con la función de encapsular los diferentes algoritmos de análisis para que puedan funcionar sobre la plataforma DiVA [5]. Para ello incorpora un cliente para conectarse a un servido de cuadros (*FrameServer*) que en nuestro caso utilizará una captura de datos desde un búffer, un método de presentación de datos y añade otro para el procesado de análisis de imagen.

3.3.1. Múltiples configuraciones de partes del cuerpo en DiVA

Una vez explicado la implentación del algoritmo propuesto por [3] en C++, veáse la sección 3.2, y explicadas las características necesarias para la integración de cualquier algoritmo en DiVA, sección 3.3, en esta sección explicaremos brevemente los pasos dados para la integración del algoritmo en DiVA, hablaremos tanto del constructor como de los métodos públicos creados para la integración del algoritmo en esta plataforma.

Para **integrarlo** hemos creado una clase a la que hemos llamado *mylatentsvm*. Las partes más importantes de esta clase son el constructor, que cargará los parámetros iniciales, el método que procesa cada frame y los métodos que permiten cambiar los parámetros.

- **Constructor** *mylatentsvm::mylatentsvm()*

  En primer lugar este constructor inicializa un objeto, al que llamamos *nuevodetector* que, tal y como explicamos en la sección 3.2-Constructores y destructores, servirá para cargar los cambios en las configuraciones utilizadas para la detección que realice el usuario. A continuación inicializa un objeto, al que llamamos *detector* (ver sección 3.2-Constructores y destructores) que leerá los datos del fichero de configuraciones generado por el usuario y actualizará los parámetros de la detección según sea necesario durante la ejecución del algoritmo.

- **Método público** *void* mylatentsvm::process(IplImage *frame)*

  El procesamiento de *frames* se lleva a cabo en este método, para ello hará uso de las variables que forman parte del objeto *detector* y que contienen la información necesaria para la detección. *Process* recibe cada *frame* y realiza la detección mediante la función *cvLatentSvmDetectObjects* tal y como se explicó en la sección 3.2-Métodos.

- **Método set**

  La función **set** es la encargada de recibir las preferencias del usuario (configuraciones, umbral y solape) y modificar *detector* a fin de que se actualicen las

variables de la detección. Para ello deberá llamar a *nuevodetector.rewrite,* en primer lugar para generar el nuevo fichero y, a continuación, a *detector.update* para actualizar la información de detector.

Como ya explicamos a la hora de modificar el umbral y el solape sólo será necesario reasignar el valor de la variable.

Hemos implementado también un destructor apropiado para esta clase, así como un método para mostrar la detección final de cada frame.

Capítulo 4

Demostrador

Una vez explicadas las pautas seguidas para la implementación del algoritmo y su integración en la plataforma DiVA sólo queda desarrollar la aplicación del demostrador diseñado. Se ha diseñado una interfaz gráfica que permite comprobar la funcionalidad del algoritmo de detección de personas basado en múltiples configuraciones de partes del cuerpo [3] y configurar los parámetros de forma sencilla.

Para desarrollar la interfaz hemos hecho uso del programa *Qt Designer,* que utiliza la biblioteca Qt, de la que hablamos en el capítulo 2. A través de *Qt Designer* diseñamos la aplicación, sus títulos, iconos, logos, barras de herramientas y ventanas adicionales. Ha sido necesario crear una nueva clase, a la que hemos llamado *GUI_latent,* para poder implementar las diferentes funcionalidades de la interfaz, además *Qt Designer* genera automáticamente archivos *ui* que facilitan la interacción entre la interfaz y el código programado.

Para este trabajo hemos diseñado diferentes ventanas para la interfaz: la ventana principal, la de selección de cámara o vídeo, la de ayuda y la de información adicional.

La ventana principal contiene las conexiones necesarias entre las funciones del código y los botones o acciones de la barra de herramientas. A partir de ella podemos conectar nuestro algoritmo a diferentes cámaras o servidores para recibir la secuencia a analizar, iniciar o detener el algoritmo, cambiar los parámetros de detección o acceder a la ventana de ayuda, información etc., explicaremos con más detalle la funcionalidad de la ventana principal a continuación. La ventana de cámaras nos permite editar la selección de cámaras. Por su parte, la ventana de ayuda e información proporcionan información sobre los diferentes parámetros modificables del algoritmo e información sobre el desarrollo del algoritmo.

4.1. Funcionalidad de la interfaz gráfica

En esta sección presentaremos la funcionalidad de la interfaz gráfica diseñada para el algoritmo explicado en el capítulo 2 cuya implementación en C++ se ha visto también en el capítulo 3 de este trabajo. Centraremos la explicación en la ventana principal y la ventana de cámaras puesto que contienen la funcionalidad necesaria para ejecutar el algoritmo. En la figura 4.1 se muestran las diferentes partes de la ventana principal.

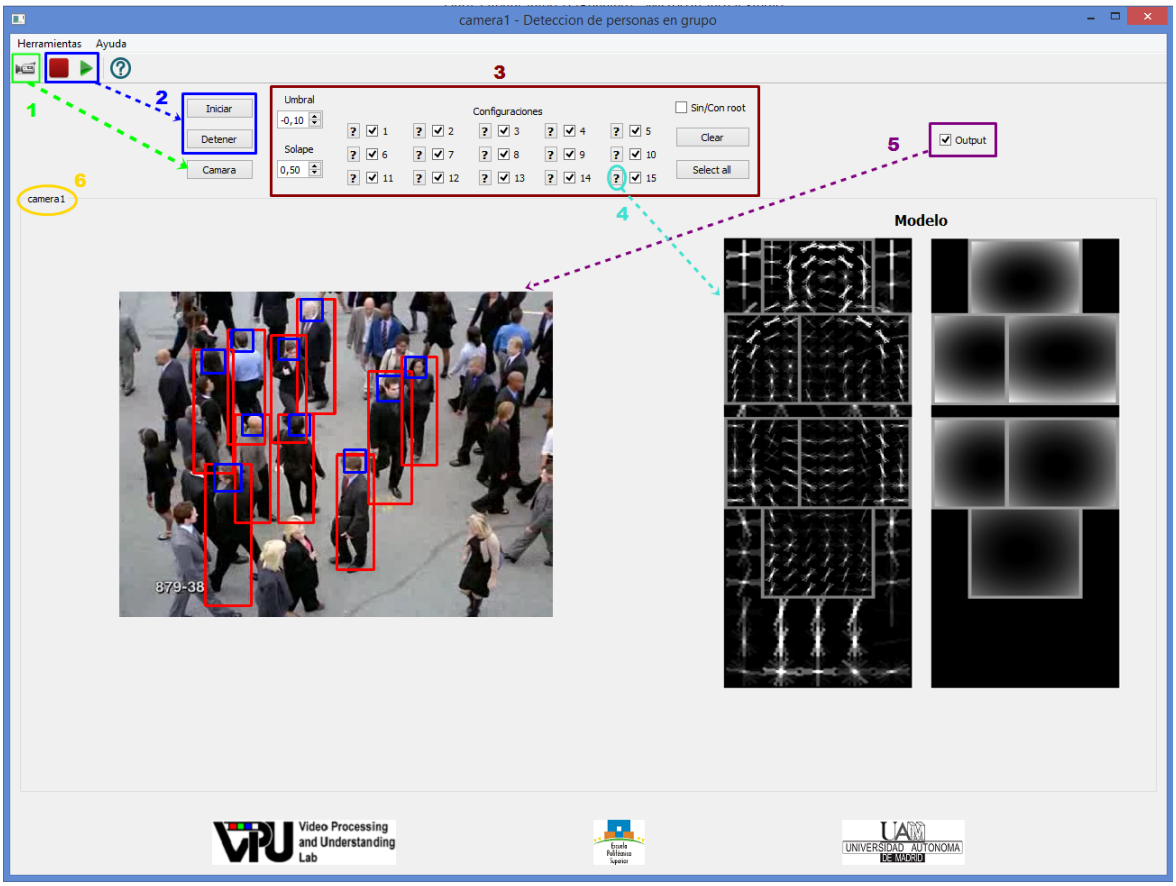


Figura 4.1: Interfaz gráfica para el algoritmo de múltiples configuraciones [3].

Atendiendo a la numeración de la imagen, la funcionalidad de cada parte es la siguiente:

1. **Selección de la cámara.**

Estos botones dan acceso a la ventana de cámaras, mostrada en la figura 4.2, que permite seleccionar una dirección IP y un puerto para la cámara a la que queremos conectarnos. Además permite guardar diferentes cámaras asignando diferentes nombres e incluso la edición de cámaras ya guardadas.

Figura 4.2: Ventana para la interfaz de selección de cámara.

2. **Inicio y detención del algoritmo**

Comenzaremos a ejecutar el algoritmo una vez que pulsemos el botón de inicio, del mismo modo podremos detener el algoritmo pulsando detener. En la figura 4.3 se ve en detalle la apariencia de esta parte de la ventana.

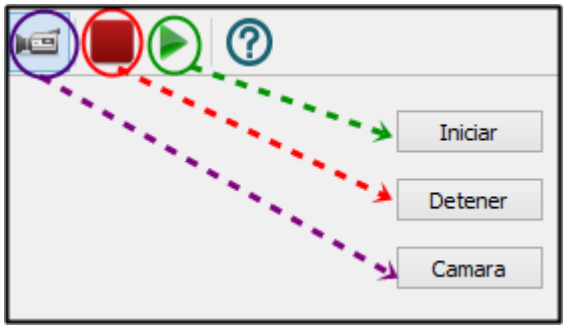


Figura 4.3: Detalle de los botones de inicio y detención del algoritmo.

3. **Selección de parámetros**

En esta sección podemos modificar los parámetros configurables del algoritmo, como se puede apreciar en la figura 4.4 los parámetros que se pueden modificar son el umbral, el solape y las diferentes configuraciones, parámetros que a su vez

son explicados brevemente en la ventana de ayuda. Para estas últimas hemos escogido un diseño que permite escoger hasta un máximo de 15 configuraciones ya que, como se ya explicamos en la sección 2.2.4 del capítulo 2, las 15 primeras configuraciones sólo se diferencian de las 15 siguientes en que contienen además el filtro *root*. Por esta misma razón hemos incluido el botón *Sin/Con root* para seleccionar cualquiera del conjunto de configuraciones sin *root* o con él, en función de si el botón este desmarcado o marcado, respectivamente. Cabe destacar que el algoritmo se inicia con las 30 configuraciones activadas y es a partir del momento que el usuario selecciona, o de-selecciona, alguna de las configuraciones que se empieza a trabajar de la forma explicada en este apartado.

De igual modo, el botón *Clear* sirve para borrar todas las selecciones, en ese momento el algoritmo seguirá ejecutándose usando la configuración básica explicada en la sección 2.2.3. Por su parte, *Select all* marcará todas las configuraciones y el algoritmo seguirá ejecutándose con las 15 configuraciones que correspondan, en función de hemos marcado o no la casilla de *root*.

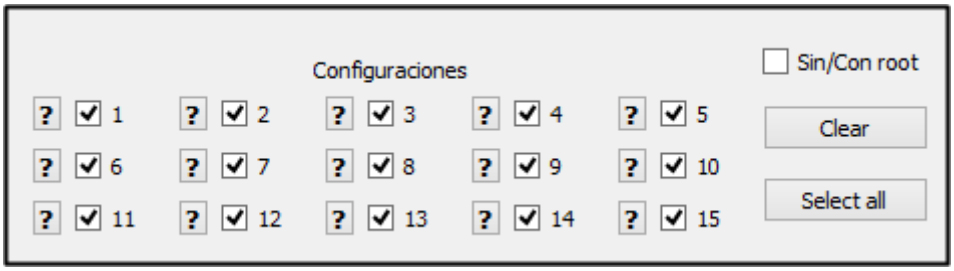


Figura 4.4: Detalle de la sección de parámetros.

Los iconos de interrogación serán explicados en el siguiente punto.

4. **Ayuda de configuraciones**

   Pulsando estos botones obtenemos una imagen de cómo es la configuración correspondiente, de este modo se pretende que el usuario pueda hacerse una idea de qué configuración puede funcionar mejor para cada escenario. A modo de ejemplo en la imagen se muestra la ayuda obtenida sobre la configuración número 8.

5. **Output**

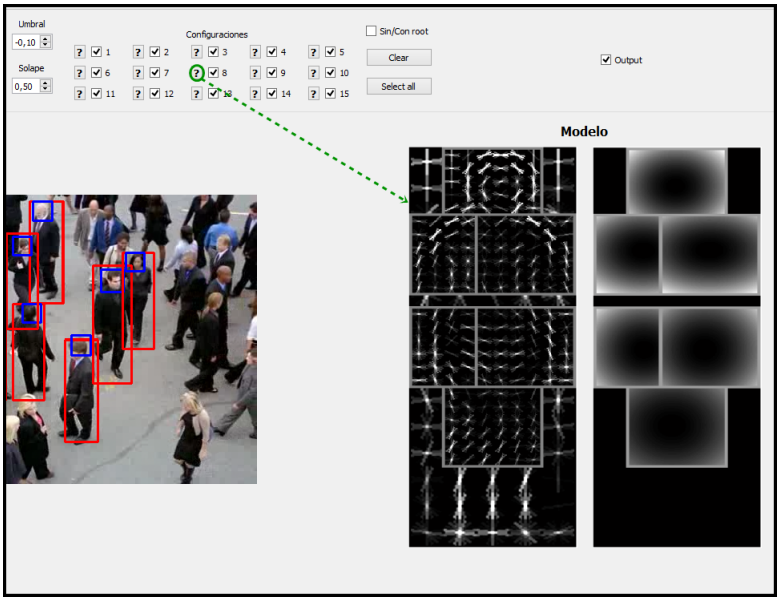


Figura 4.5: Detalle de la ayuda de configuraciones

Es importante destacar que no podremos visualizar el resultado del algoritmo hasta que esta opción esté marcada, el algoritmo seguirá en ejecución y mostrará un mensaje de error para recordar que se debe marcar la opción *Output*. Cuando desmarcamos esta opción se borrará cualquier imagen, incluidas las de ayuda de configuraciones que se esté mostrado en el momento.

6. **Nombre de la cámara seleccionada**

   Da información acerca de la cámara a la que estamos conectados en ese momento. Este nombre está asignado por el usuario en el momento de conectarse a la cámara.

4.2. Caso de estudio

Una vez explicada la funcionalidad de la interfaz vamos a mostrar un par de ejemplos visuales que ilustran el funcionamiento del algoritmo. Como observamos en las figuras 4.6 y 4.7 usando como referencia el mismo frame, con el mismo umbral y solape se detectan más personas usando algunas de las diferentes configuraciones propuestas por el algoritmo de [3].

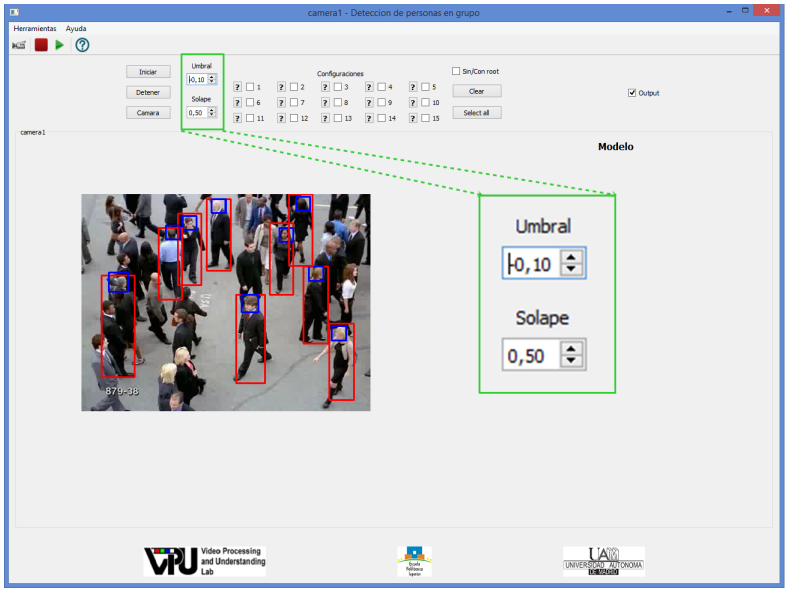


Figura 4.6: Resultado empleando la configuración básica. Misma configuración que la usada en el algoritmo DTDP [1]

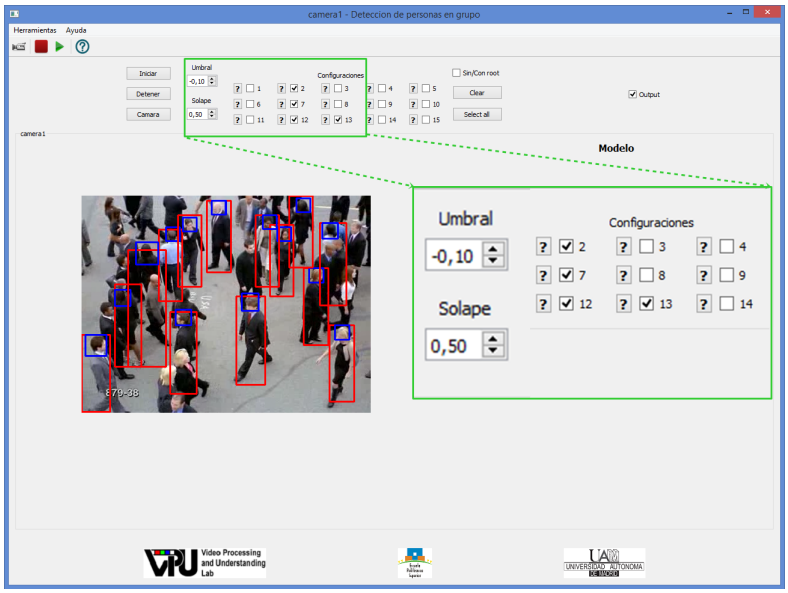


Figura 4.7: Resultado empleando las las configuraciones 2, 7, 12 y 13 del algoritmo base [3].

De la misma forma, se puede observar que se obtienen mejores resultados en otra secuencia de vídeo a pesar de que el grupo de personas se ve desde una perspectiva más alejada y las personas se ven más pequeñas y juntas entre ellas. De igual modo

se ha probado se ha probado el mismo frame con el mismo umbral y nivel de solape tanto para la configuración básica (figura 4.8) como con todas las configuraciones diseñadas (figura 4.9)

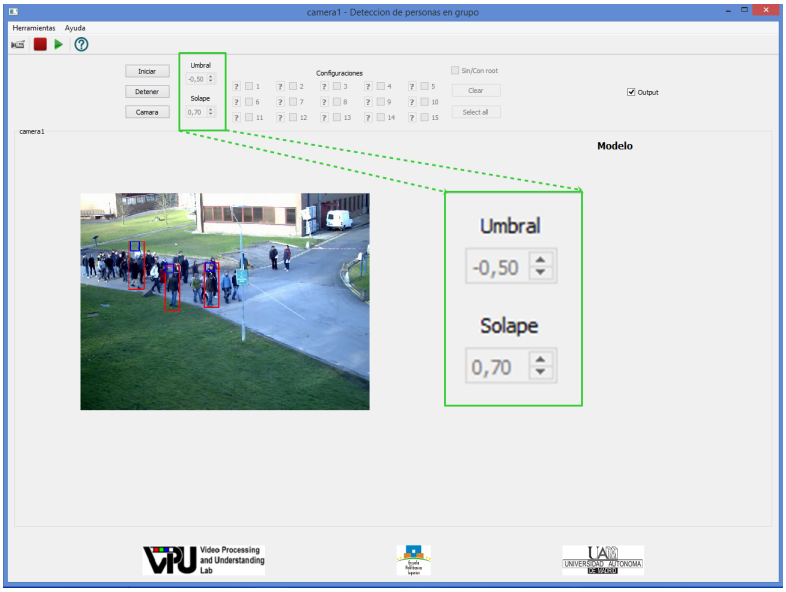


Figura 4.8: Resultado empleando la configuración básica para la segunda secuencia de vídeo. Misma configuración que la usada en el algoritmo DTDP [1]

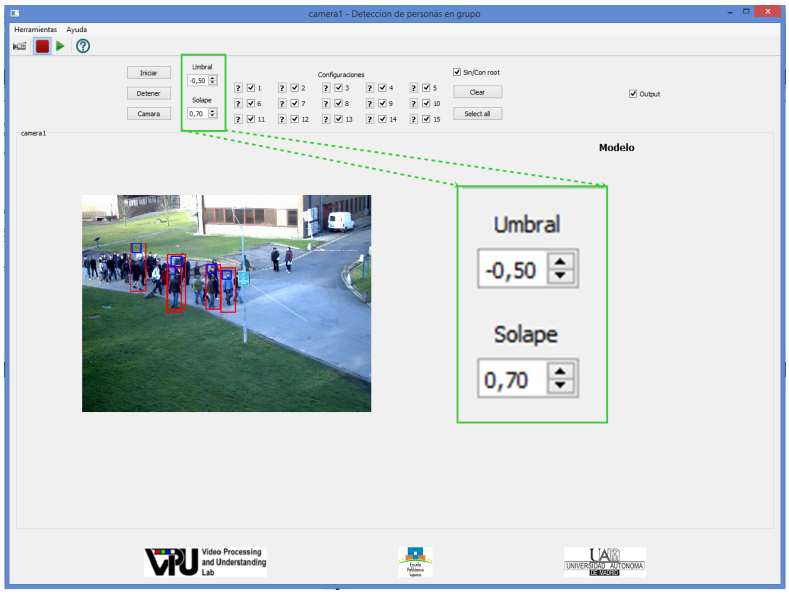


Figura 4.9: Resultado empleando las diferentes configuraciones para otra secuencia de vídeo

Capítulo 5

Conclusiones y trabajo futuro

Este trabajo tenía como objetivo la adaptación de un algoritmo de detección de personas en grupo, así como su encapsulación en una plataforma que permitiese la conexión con diferentes cámaras o vídeos y el desarrollo de una interfaz que facilitase la interacción persona-algoritmo para poder comprobar de forma sencilla la funcionalidad del mismo.

Para conseguir este objetivo se ha realizado un estudio del estado del arte, capítulo 2, del algoritmo propuesto por [3], explicado en la sección 2.2.4, así como del algoritmo original DTDP. A partir de este estudio se han aprendido los aspectos más importantes del algoritmo, incluyendo tanto sus ventajas como limitaciones.

En el capítulo 3 se han explicado los pasos dados para su implementación en C++ y su integración en la plataforma DiVA. Finalmente, en el capítulo 4, hemos comentado la funcionalidad de la interfaz gráfica diseñada.

A la vista de los resultados podemos concluir que se han cumplido los objetivos de esta trabajo, tanto la implementación el algoritmo como su integración en una interfaz gráfica que permite al usuario interaccionar de forma sencilla y comprobar su funcionalidad. Sin embargo, es evidente la necesidad de seguir trabajando en algoritmos de detección de personas en grupos ya que se trata de una línea de investigación compleja. En concreto para este trabajo y para el algoritmo en el que se basa se proponen unas líneas a seguir como parte del trabajo futuro.

El algoritmo en el que se basa este trabajo es un algoritmo lento, razón por la cual no ha sido posible su ejecución sobre vídeos en tiempo real, es por eso que una de las líneas de investigación propuesta como trabajo futuro es el desarrollo de un algoritmo más rápido, limitando la zona de búsqueda espacial ya que se trata de un algoritmo de búsqueda exhaustiva, otra opción puede ser limitar el rango de escalas a analizar de forma automática. Proponemos además la implementación del algoritmo

sin necesidad de usar un fichero de configuraciones, de esta forma se podría disminuir ligeramente el tiempo de ejecución y ayudaría de cara a conseguir la siguiente mejora propuesta.

Como mejora del algoritmo también se propone la creación de otras configuraciones que permitan un modelo de persona más genérico, así como permitir que el propio usuario determine el tipo de configuración que desea, es decir, creando diferentes configuraciones con las partes del cuerpo que quiera.

Aunque en este trabajo no se han analizado este tipo de algoritmos se propone el estudio de algoritmos de seguimiento o *tracking,* que mejoren la detección. Este tipo de algoritmos son más robustos ante oclusiones es por eso que se cree probable que combinados con el algoritmo añadan robustez y mejoren la detección.

Bibliografía

[1] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32(9), pp. 1627–1645, 2010.

[2] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. of CVPR*, pp. 886–893, 2005.

[3] A. Garcia-Martin, R. H. Evangelio, and T. Sikora, "Multi-configurations for part-based person detectors," in *Proc. of ECCV*, 2014.

[4] OpenCV, "http://opencv.org/,"

[5] J. C. SanMiguel and J. M. Martinez, "Strategies for object segmtation, detection and tracking in complex environments for event detection in video surveillance and monitoring.," tech. rep., DiVA Documentation D1.2v1 TEC 2011-25995 EventVideo, 2012-2014.

[6] Qt-Project, "http://www.qt.io/developers/,"

[7] A. G. Martín, *Contributions to robust people detection in video-surveillance*. PhD thesis, Universidad Autónoma de Madrid, 2013.

[8] D. Gerónimo, A. M. López, A. D. Sappa, and T. Graf, "Survey of pedestrian detection for advanced driver assistance systems," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32(7), pp. 1239–1258, 2010.

[9] A. Broggi, M. Bertozzi, A. Fascioli, and M. Sechi, "Shape-based pedestrian detection," in *Proceedings of the IEEE Intelligent Vehicles Symposium*, pp. 215–220, Citeseer, 2000.

[10] D. M. Gavrila and S. Munder, "Multi-cue pedestrian detection and tracking from a moving vehicle," *International Journal of Computer Vision*, vol. 73(1), pp. 41–59, 2007.

[11] D. M. Gavrila, "Pedestrian detection from a moving vehicle," in *Proc. of ECCV*, pp. 37–49, 2000.

[12] A. Andriyenko, K. Schindler, and S. Roth, "Discrete-continuous optimization for multi-target tracking," in *Proc. of CVPR*, pp. 1926–1933, 2012.

[13] B. Wu and R. Nevatia, "Detection and tracking of multiple, partially occluded humans by bayesian combination of edgelet based part detectors," *International Journal of Computer Vision*, vol. 75(2), pp. 247–266, 2007.

[14] S. Tang, M. Andriluka, and B. Schiele, "Detection and tracking of occluded people," *International Journal of Computer Vision*, vol. 110, pp. 58–69, 2014.

[15] A. Garcia-Martin, A. Cavallaro, and J. M. Martinez, "People-background segmentation with unequal error cost," in *Proc. of ICIP*, pp. 157–160, 2012.

[16] S. Kullback and R. A. Leibler, "On information and sufficiency," *The Annals of Mathematical Statistics*, vol. 22(1), pp. 79–86, 1951.